Securing the Weakest Link

Sean Barnum, Cigital, Inc. [vita[3]]
Michael Gegick, Cigital, Inc. [vita[4]]

Copyright © 2005 Cigital, Inc.

2005-09-19

L4 / D/P, L[5]

Attackers are more likely to attack a weak spot in a software system than to penetrate a heavily fortified component. For example, some cryptographic algorithms can take many years to break, so attackers are not likely to attack encrypted information communicated in a network. Instead, the endpoints of communication (e.g., servers) may be much easier to attack. Knowing when the weak spots of a software application have been fortified can indicate to a software vendor whether the application is secure enough to be released.

Detailed Description Excerpts

According to Viega and McGraw [Viega 02] in Chapter 5, "Guiding Principles for Software Security," in "Principle 1: Secure the Weakest Link" from pages 93-96:[9]

> Security practitioners often point out that security is a chain; and just as a chain is only as strong as the weakest link, a software security system is only as secure as its weakest component. Bad guys will attack the weakest parts of your system because they are the parts most likely to be easily broken. (Often, the weakest part of your system will be administrators, users or tech support people who fall prey to social engineering.)
>
> It's probably no surprise to you that attackers tend to go after low-hanging fruit. If a malicious hacker targets your system for whatever reason, they're going to follow the path of least resistance. That means they'll try to attack the parts of the system that look the weakest, and not the parts that look the strongest. (Of course even if they spend an equal effort on all parts of your system, they're far more likely to find exploitable problems in the parts of your system most in need of help.)
>
> A similar sort of logic pervades the physical security world. There's generally more money in a bank than a convenience store, but which one is more likely to be held up? The convenience store, because banks tend to have much stronger security precautions. Convenience stores are thus a much easier target. Of course the payoff for successfully robbing a convenience store is much lower than knocking off a bank; but it is probably a lot easier to get away from the convenience store crime scene.
>
> So to stretch our analogy a bit, you want to look for and better defend the convenience stores in your software system.
>
> Consider cryptography. Cryptography is seldom the weakest part of a software system. Even if a system uses SSL-1 with 512-bit RSA keys and 40-bit RC4 keys (which is, by the way, considered an incredibly weak system all around) an attacker can probably find much easier ways to break the system than attacking the crypto. Even though this system is definitely breakable through a concerted crypto attack, successfully carrying out the attack requires a large computational effort and some knowledge of cryptography.
>
> Let's say the bad guy in question wants access to secret data being sent from point A to point B over the network (traffic protected by SSL-1). A clever attacker will target one of the endpoints, try to find a flaw like a buffer overflow, and then look at the data before it gets encrypted, or after it gets decrypted. Attacking the data while encrypted is just too much work. All the cryptography in the world can't help you if there's an exploitable buffer overflow, and buffer overflows abound in code written in C.

3. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/35-BSI.html (Barnum, Sean)
4. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/345-BSI.html (Gegick, Michael)
9. All rights reserved. It is reprinted with permission from Addison-Wesley Professional.

For this reason, while cryptographic key lengths can certainly have an impact on the security of a system, they aren't all that important in most systems, where there exist much bigger and more obvious targets.

For similar reasons, attackers don't attack a firewall unless there's a well-known vulnerability in the firewall itself (something all too common, unfortunately). Instead, they'll try to break the applications that are visible through the firewall, since these applications tend to be much easier targets. New development tricks and protocols like SOAP, a system for tunneling traffic through port 80, make our observation even more relevant. It's not about the firewall; it's about what is listening on the other side of the firewall.

Identifying the weakest component of a system falls directly out of a good risk analysis. Given good risk analysis data, addressing the most serious risk first, instead of a risk that may be easiest to mitigate, is always prudent. Security resources should be doled out according to risk. Deal with one or two major problems, and move on to the remaining ones in order of severity.

Of course, this strategy can be applied forever, since 100% security is never attainable. There is a clear need for some stopping point. It is okay to stop addressing risks when all components appear to be within the threshold of acceptable risk. The notion of acceptability depends on the business proposition, of course.

Sometimes it's not the software that is the weakest link in your system; sometimes it's the surrounding infrastructure. For example, consider social engineering, an attack in which a bad guy uses social manipulation to break into a system. In a typical scenario, a service center will get a call from a sincere sounding user, who will talk the service professional out of a password that should never be given away. This sort of attack is easy to carry out, because customer service representatives don't like to deal with stress. If they are faced with a customer who seems to be really mad about not being able to get into their account, they may not want to aggravate the situation by asking questions to authenticate the remote user. They will instead be tempted just to change the password to something new and be done with it.

To do this right, the representative should verify that the caller is in fact the user in question who needs a password change. Even if they do ask questions to authenticate the person on the other end of the phone, what are they going to ask? Birthdate? Social Security number? Mother's maiden name? All of that information is easy for a bad guy to get if they know their target. This problem is a common one, and is incredibly difficult to solve.

One good strategy is to limit the capabilities of technical support as much as possible (remember, less functionality means less security exposure). For example, you might choose to make it impossible for a user to change a password. If a user forgets their password, then the solution is to create another account. Of course, that particular example is not always an appropriate solution, since it is a real inconvenience for users. Relying on caller ID is a better scheme, but that doesn't always work either. That is, caller ID isn't available everywhere. Moreover, perhaps the user is on the road, or the attacker can convince a customer service representative that they are the user on the road.

The following somewhat elaborate scheme presents a reasonable solution. Before deploying the system, a large list of questions is composed (say, no fewer than 400 questions). Each question should be generic enough that any one person should be able to answer it. However, the answer to any single question should be pretty difficult to guess (unless you are the right person). When the user creates an account, we select 20 questions from the list, and ask the user to answer 6 of them that the user has answers for, and is most likely to give the same answer if asked again in two years.

Here are some sample questions:

- What is the name of the celebrity you think you most resemble, and the one you would most like to resemble?
- What was your most satisfying accomplishment in your high school years?
- List the first names of any significant others you had in high school.

- Whose birth was the first birth that was significant to you, be it a person or animal?
- Who is the person in whom you were most interested to whom you never expressed your interest (your biggest secret crush)?

When someone forgets their password and calls technical support, technical support refers the user to a web page (that's all they are given the power to do). The user is provided with three questions from the list of six, and must answer two correctly. If they answer two correctly, then we do the following:

- Give them a list of 10 questions, and ask them to answer three more.
- Let them set a new password.

We should probably only allow a user to authenticate in this way a small handful of times (say, three).

The result of this scheme is that users can get done what they need to get done when they forget their passwords, but tech support is protected from social engineering attacks. We're thus fixing the weakest link in a system.

All of our asides aside, good security practice dictates an approach that identifies and strengthens weak links until an acceptable level of risk is achieved.

According to Schneier [Schneier 00] in "Security Processes":

Secure the Weakest Link. Spend your security budget securing the biggest problems and the largest vulnerabilities. Too often, computer security measures are like planting an enormous stake in the ground and hoping the enemy runs right into it. Try to build a broad palisade.

Further Reading

Crafting malicious input is one method of circumventing strong encryption or user authentication. For example, an attacker need only inject a simple SQL command in a web form to obtain a list of usernames that are encrypted in a database. The failure to sanitize user input that contains SQL commands represents a weak link that attackers often prey on. Other examples of malformed input include the injection of shell commands that are executed with root privileges and cross-site scripting attacks, in which script code is injected in HTML pages that can reveal a user's cookie information. Attackers don't stop at text entry fields; they can also manipulate protocols (e.g., HTTP GET requests and payload information) to wreak havoc on a victim machine. Preventing users from being able to enter arbitrary data into an application is crucial for providing good security.

A *black list* [Hoglund 04] can be created that contains possible forms of malicious input for a data entry field that developers can check against in their code. However, determining all variations of unsafe input is infeasible because of the sheer number of possible exploits an attacker can employ. Instead, developers may code according to what a *white list* [Hoglund 04] defines as well-formed input for a given input field. Good requirements engineering makes it possible to know exactly what input is expected so the proper security checks can be put in place. Securing the code that interfaces with the user will indubitably decrease the likelihood of a successful attack.

References

[Hoglund 04]	Hoglund, Greg & McGraw, Gary. *Exploiting Software: How to Break Code.* Boston, MA: Addison-Wesley, 2004.
[Schneier 00]	Schneier, Bruce. "The Process of Security[14]." *Information Security Magazine,* April, 2000.
[Viega 02]	Viega, John & McGraw, Gary. *Building Secure Software: How to Avoid Security Problems the Right Way*. Boston, MA: Addison-Wesley, 2002.

Cigital, Inc. Copyright

Copyright © Cigital, Inc. 2005-2007. Cigital retains copyrights to this material.

Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

For information regarding external or commercial use of copyrighted materials owned by Cigital, including information about "Fair Use," contact Cigital at copyright@cigital.com[1].

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. mailto:copyright@cigital.com